DATABASE ABSTRACTIONS = DATA ABSTRACTIONS + SEMANTIC INTEGRITY CONSTRAINTS ABSTRACTIONS

Heng-Li Yang

楊　亨　利*

摘　要

傳統資料庫設計技術集中在結構的設計與效率的考慮。然而，一個已是第四甚或第五正規化的關連式資料庫仍代表很少的資料語意。一個資料庫的概廓應包含「結構」的部份及「語意完整性限制」部份。同樣地，「資料庫抽象化」也應包含「資料抽象化」及「語意完整性限制抽象化」。「資料抽象化」用以將真實世界的「結構」部分概念化、抽象化。而本文提出新的抽象化觀念以對「語意完整性限制」造模。這將減少必須明顯及完全表示出來的語意完整性限制，並有助於語意完整性限制的組織與管理。

關鍵詞：語意完整性限制；資料抽象化；語意完整性限制抽象化；資料庫設計；增強式資料庫

Abstract

Traditional datadase design techniques focus on structure and efficiency. However, a relational database in fourth or fifth normal form may still represent little of the data semantics. A database schema should consist of both structures and semantic integrity constraints (SICs). Similarly, database abstractions should also include data abstractions and semantic integrity constraints abstractions. Data abstractions have been applied to conceptualize the structure part of the real world. This paper proposes new abstractions to model the semantic integrity constraints. It would reduce the number of SICs, which must be represented explicitly and fully, and help the SIC organization and management.

Keywords: Semantic Integrity Constraints; Data Abstractions; SIC Abstractions; Database Design, Enhanced Databases

1 Introduction

Traditionally, a database schema includes only the structural specifications. For

* 作者爲本校資管系副教授

example, if the relational model is adopted, a database designer would only specify the names of tables, the names and widths of columns in each table, etc. The data semantics are missing. *Semantic inteqrity* is concerned with the logical meaning (i.e., the intension) of stored data and preserving the correctness of database contents even though users and application programs try to modify it incorrectly [11,10]. A database schema should consist of structures and *semantic integrity constraints* (hereafter abbreviated as SICs) [46,12,50]. The structure part tells us relatively little information other than the basic structure – what elementary items of data are grouped into what larger units; but the SIC part provides information about all allowable occurrences – current and future [29]. These SICs express *data integrity semantics,* that is, the part of the meaning of stored data needed to determine correctness. They are invariant restrictions on the static states of the stored data and the state transitions caused by the primitive operations: insertion, deletion, or update. They express what is and is not allowed in the part of the universe that is represented by the stored data.

An *abstraction* is a simplified description of a system that suppresses specific details while emphasizing those pertinent to the problem. *Database abstractions* should consist of *data abstractions* and *SIC abstractions* that are proposed in this paper. Data abstraction concerns abstraction applied to the structure part. SIC abstraction concerns abstraction applied to semantic integrity constraints.

There are a lot of benefits, such as enhanced integrity, easier query formulation, and database design localization, can be received by incorporating data abstractions (e.g., [6,25]). It is therefore an important construct in most of semantic data models [18]. However, in literature, researchers might have different perspectives or use the same terminology to imply different things [16], which causes confusion.

Traditionally, there has been little interaction between researchers in the areas of AI and databases. Recent trends have shown that techniques or ideas developed in each of these areas may be adapted for use in the other (references, e.g., [47,20,21,48,27,4,1,2,39,23,32]). There are several ways to integrate expert systems and database systems. Researchers [21,1] have classified these integrated systems into three classes: (1) intelligent databases (enhanced databases); (2) enhanced expert systems; (3) inter-system communication (coupling of existing expert systems and databases).

There are some ways to enhance databases by expert system capabilities or methods, e.g., deductive database approach [14,22], incorporating more semantic integrity constraints or rules into database [29,50,38,40,37,49]. One major related research for incorporating semantic integrity constraints or rules is on the new database type: object-oriented databases (e.g., [15,3,19]). The term "database" in this paper in fact represents an enhanced relational database. This research retains the traditional relational model because of its popularity, but proposes to include the necessary SICs in addition to relation structures in a relational schema. Under this architecture, a database management system should have an integrity maintenance subsystem to enforce SICs. A more closely related research is on active relational databases [38,40,37,49,28,7]. However, the aspects of SIC design, organization and management have been seldom discussed. In addition, their SIC representations are not precise.

The objectives of this paper is twofold. The first objective of this paper is to review and synthesize data abstractions from the literature to clarify the confusion. The second objective is to explore the possibilities of applying the important abstraction construct to the SIC specifications — SIC abstractions. Benefits of incorporating SIC abstractions are such as, facilitating SIC design, organization and management.

The paper organization is as follows. The remainder of this section provides an introduction to the *SIC Representation model.* Section 2 performs the review and clarification of data abstractions. Section 3 proposes the new SIC abstractions. Applying SIC abstractions, Section 4 introduces a new idea to represent the SICs in a database schema. This idea will apply SIC abstractions and the SIC Representation model. It would help not only incorporate SICs in a database schema, but also represent and manage them in the populated database. Its subsection 4.2 gives some examples. Section 5 concludes this paper.

1.1 SIC Representation Model

This subsection only gives some introduction to a new SIC representation form. The details of this *SIC Representation model* are described in [50,51,42].

In literature, there are a number of researchers proposing languages to represent SICs (see [50,51] for good review). However, their representations are not precise. Su & Raschid [43] and Shepherd & Kerschberg [34] suggest that "constraints" must explicitly specify both declarative semantics as well as process oriented or operational semantics. *Declarative semantics* correspond to logical formulas describing relationships between objects in a database. The information needed to check that these relationships are true or to maintain these relationships is the *operational semantics.* Therefore, any single SIC representation in the schema should contain the following components in order to express its features precisely [50]:

1. **Certainty Factor:** measures the certainty of the SIC.
2. **Object:** represents the data object to which the SIC applies.

3. **Operation Type:** specifies the type of database manipulation operation (insertion, deletion, or update) to which the SIC is applicable.
4. **Precondition:** specifies the general state of the database that makes the SIC relevant.
5. **Predicate:** is an assertion specifying something that must be true about the object after the operation.
6. **Violation Action:** specifies how the system to behave if the predicate is false when the SIC is checked.

Each SIC is then represented as:

SIC-Name

CERTAINTY	**F** (Certainty Factor)
FOR	**O** (Object)
ON	**T** (Operation Type)
IF	**C** (Precondition)
ASSERT	**P** (Predicate)
ELSE	**A** (Violation Action)

In terms of the usual production rule syntax, the above whole statement (except for *SIC-Name*) is interpreted as:

```
with certainty F
IF (O,T,C)      THEN
                     (IF NOT P THEN A)
```

SIC-Name is used as an identifier that conveys some meaning of a SIC. It is not essential to the Representation model.

The following example, named as *project_employee_minimum_salary* for convenience, will be used for illustration. Suppose that there is an *Emplyoyee* entity, and a *Work_for* relationship in the Entity-Relationship (E-R) model. A SIC might state *"if an employee works for any project, then his (her) salary should be greater than $10,000"*. By applying the algorithms proposed in [50,51], this **general SIC** can be decomposed into several **sub-SICs** represented in the above format. Each sub-SIC is operation-dependent and only relevant to a single object. In addition to the related sub-SICs on the update of *Work_for's* primary key, two sub-SICs are:

- one for *Employee.Salary* on update;
- one for *Work_for* on insertion

The first one may be represented as the following.

Employee.Salary-U-RshipDepEntVal-(Work_for)

CERTAINTY	certain
FOR	Employee.Salary
ON	update
IF	∃ Work_for, rship_occ_part (Work_for, "Employee", Employee)
ASSERT	Employee.Salary > 10000
ELSE	reject

Interpretation: The first line is the SIC name. It indicates that this is a SIC for *Employee.Salary* on update and it is a *"RshipDepEntVal"* type because it asserts that the existence of a relationship (*Work_for*) depends on the value of an entity attribute (*Employee.Salary*). The *rship_occ_part* is an assertion that stands for "relationship occurrence participant". In this example, it is used to specify the *Work_for* occurrence in which the currently checked *Employee* occurrence participates with the entity type, "Employee". This SIC states that with 100% certainty, when an *Employee.Salary* occurrence is to be updated, if the *Employee* participates in at least one *Work_for*, his or her *Salary* must be greater than $10,000. Otherwise, the update operation is rejected. Note that the database designer might choose "propagate (delete (Work_for))" as the violation action. In that case, the propagation action might imply that if the organization could not afford the minimum salary for an employee, it could not require that the employee be associated with any project (so the current *Work_for* occurrence must also be deleted).

Unfortunately, a complete set of SICs are not easily specified because of the complexities of data semantics. Furthermore, the number of SIC specifications represented in the above format might become so huge that a database designer could not specify them all.

2 Data Abstractions

In the following, three kinds of data abstractions are reviewed and clarified: inclusion, aggregation, and association [50]. They are discussed here for two reasons as follows.

- First, in literature, these three have been the most common abstractions. However, their definitions, especially for the aggregation and association, are confusing. For example, the association abstraction is sometimes referred to as membership, grouping, partitioning, or cover aggregation [31].

- Second, the proposed new SIC abstractions in Section 3 are similar to these three abstractions.

Other data abstractions (e.g., *materialization* [17]) or semantic relationships (e.g., case relationships, antonyms [41] are not discussed here.

2.1 Inclusion Abstraction

The **inclusion** abstraction concept [16] encompasses **classification, generalization, and specialization.** Classification is a form of abstraction in which a type is defined as a collection of occurrences with common properties. Specialization occurs when every occurrence of a type is also an occurrence of another type. Specialization is indicated by the term, *is_a*, that is, *S is_a G*, where *S* is a subtype and *G* is a supertype. It is possible to have generalization or subset hierarchies [44]. A generalization hierarchy occurs when a type is union of non-overlapping subtypes. A subset hierarchy occurs when a type is union of possibly overlapping subtypes.

Property inheritance, which means that attributes, associated relationship and imposed SICs of the super-type are inherited by each subtype (or a type's are inherited by its occurrences), is the most important characteristic of the inclusion abstraction. In the case of the classification abstraction (related occurrences to types), the property inheritance principle has been enforced traditionally by any DBMS. This paper assumes that DBMS (either in the E-R model or the relational model) would also automatically implement property inheritance for the specialization abstraction. That is, the principle of redundancy minimization, i.e., properties that can be inherited from another entity type via an *is_a* relationship should not be stored explicitly [16], has been followed.

Otherwise, the inherited attributes, relationships and imposed SICs would need to be explicitly and redundantly stored for each specific subtype and other SICs would be required to assure that these properties have been really "inherited" !

2.2 Aggregation Abstraction

Aggregation is an abstraction that allows a relationship between objects (i.e., attributes, entities, relationships) to be considered as a higher level object [35]. The term, *component_of*, is used to indicate the aggregation, that is , *C component_of A*, where *C* is a component and *A* is an aggregate.

This paper classifies aggregation abstractions in the E-R model into five types as follow. Note that the classification here is more complete and clearer compared to the discussions about aggregations in literature [6,25,18,16,35,30,33,36,45].

1. *Attribute aggregation* is the abstraction in which an attribute may be defined as the aggregation of attributes.

2. *Simple entity aggregation* is the abstraction in which an entity is defined by aggregation of attributes. This is the traditional entity concept.

3. *Complex entity aggregation* is the abstraction in which an entity is defined by aggregation of attributes, other independent entities, and probably sets. For example, an entity type *Department* could be thought of as an aggregation of its *Name, Budget* (both are attributes), *Manager* (an independent entity), and *Employee* (a set). The aggregate object does not really "own" them as components. This is an abstraction used in a semantic data model, SHM+[5]. One should note that in SHM+, there is no "relationship" construct, its aggregation abstraction indeed implies a number of implicit "relationships" between the aggregate object and its components.

4. *Relationship aggregation* is the abstraction in which a "relationship" is obtained by aggregation of entities and some attributes. For example, a relationship type *Reserve* could be thought of as the aggregation of *Person, Hotel,* and *Time.* It is just another way to represent a relationship.

5. *Composite entity aggregation* is the abstraction in which an entity contains

other dependent entities and some attributes as real components. The aggregate entity "owns" these other entities [24]. That is, the existence of these other entities is dependent on their aggregate and are owned by exactly one aggregate. Although some researchers argue that in aggregation the inheritance is upwards [5,26,31], this paper arguse that it is probably more suitable to state that the aggregate "owns" components and components "own" their attributes, so the aggregate "owns", rather than "inherits", components' attributes. For example, we may state *"a car owns engine.weight, engine.brand, and brake.brand, etc."*.

2.3 Association Abstraction

Association is the abstraction in which a collection of member objects is considered as a higher level (more abstract) set object [5]. The term *member_of* is used to indicate the association, that is, *M member_of S,* where *M* is a member and *S* is a set. Brodie [5] states "as with aggregation, the inheritance goes upward" and some researchers (e.g., [26]) even state that association may support both upward and downward inheritance. This paper takes the position that there is no inheritance in association. That is, set is distinguished from type and a "set" in association is considered to represent a pure mathematical set.

Before Brodie mentioned association, dos Santos et al. [9] had already proposed a useful data abstraction **"correspondence"**, which was later referred to by Furtado and Neuhold [13] as "grouping". Grouping is more general than association. It creates a group of sets, i.e., grouping is an abstraction that defines a new entity type in which each occurrence is a set formed from a collection of occurrences of the source entity type.

According to the correspondence idea in [9], a group of sets is formed by an *indexing* mechanism. Applying the idea of correspondence, This paper classifies association abstractions in the E-R model into three types as follow. Note that the classification here is more complete and clearer compared to the discussions about association in literature [6,25,18,16,30,33,36,45,5,9].

1. **Natural Set Association:** A set, is defined to contain all occurrences of an entity *M* type. Classification is the indexing mechanism to form a set. For example, we have: *"Employee member_of Employees"* where *Employees* is a set consisting of all *Employee* occurrences in the employee type.

If these sets only have attributes that are derived from those of their members, their explicit representations may be redundant and may not be efficient after considering the enforcement of SICs. However, there may be *a priori* attributes, for example, *Employees.Representative.* In the whole database, there are a number of such sets, e.g., *Employees, Departments.* Since they may have different derived attributes and *a priori* attributes, they form different entity set types containing a single occurrence, respectively. This accociation abstraction relationship, *member_of,* should not have its own attributes and need not be explicitly represented.

2. **Indexing Derived Set Association:** This is the original correspondence abstraction discussed by dos Santos et al. [9]. The indexing mechanism can be an indexing attibute that is an attribute of the indexed entity type, or an indexing entity type that is related to the indexed entity type via an indexing relationship type. An example is *cosets of employees of the same age,* where *Employee.Age* is the indexing attribute for the indexed entity type *Employee.* If the indexing attribute is an attribute that disallows null ("unknown"), the cosets obtained by grouping would form a partition of the indexed entity type occurrences. Another example is shown in Figure 1, where *DS* is the indexing derived set (e.g., *cosets of employees who work_for the same project*), *M* (e.g., *Employee*) is the indexed entity type, *I* (e.g., *Project*) is the indexing entity type, and *R* (e.g., *Work_for*) is the indexing relationship. Grouping is a powerful abstraction. There may be more than one indexing type, which can be combined with indexing attributes as the indexing mechanism. Although we can get a group of sets from the indexing mechanism, we may only be interested in some of these (e.g., *the set of employees who work_for project p100*).

Figure 1: Grouping: Member (M), Derived Set (DS), Indexing Entity (I), Indexing Relationship (R)

In general, this association abstraction relationship, *member_of* does not have its own attributes and need not be explicitly represented. This kind of set has some attributes derived from the indexed entity type, some are defined *a priori.* Two kinds of attributes in a set are important for membership derivation: the indexing attribute (e.g., *Employee.Age*) and the primary key of the indexing entity type (e.g., *Project.Id*).

3. **Enumerated Set Association:** There is no indexing mechanism in this kind of association because the database designer does not know or does not care about the indexing entity types or attributes. The set membership can only be explicitly enumerated by the *member_of* relationship.

3 SIC Abstractions

Three kinds of SIC abstractions are proposed: aggregation, specification, and association.

SIC Aggregation Assume that *O1* and *T1* are the data objects and operation types for *SIC-1*, respectively; and *Oi* and *Ti*, where *i=2,3,...*, are the data objects and operation types for *SIC-i,* respectively. *SIC-1* (called **aggregate-SIC**) is the aggregation of other SICs (called **component-SICs**) if all the following hold:

- *O1* contains all *Oi*'s as components;
- the operation *T1* on *O1* can be conceptually thought of as the conjunction of operations *Ti* on *Oi*;
- *component-SICs* and *aggregate-SIC* are sub-SICs decomposed from the same general SIC.

The certainty factors of these *component-SICs* and *aggregate-SIC* are the same. Each *component-SIC* can exist on its own. It has its own precondition, assertion and violation action. The violation action of a *component-SIC* will be taken in the context of its own enforcement.

An *aggregate-SIC* may have its own assertions and violation action. The enforcement of an *aggregate-SIC* can be simulated by checking all of its *component-SICs* and its own assertions. If an object violates any *component-SIC*, the violation action of the *aggregate-SIC* will be taken. Thus, a special logical predicate could be used (e.g., *checkcomSIC*)[1] to refer to all its *component-SICs* by their SIC names in the *oggregate-SIC* and avoid having to write the same assertions explicitly for *O1* on *T1*.

One example is that the domain constraint on insertion of an entity can be simulated by checking the domain constraint of updating its attributes (from unknown values to some values). We would have SICs asserting not-null, value, unique, etc., for each of its attributes on update. These assertions need not be repeated for the entity on insertion.

SIC Specialization Assume that *O1* and *T1* are, respectively, the object and operation type for *SIC-1*; and *O2* and *T2* are, respectively, the object and operation type for *SIC-2*. *SIC-2* is the specialization of *SIC-1* if *O2* is a specialization (i.e., sub-type) of *O1*, and *T1* = *T2*.

The specialized SIC would inherit assertions from its parent SIC with the proper variable substitution (*O2* for *O1*). Thus, representation of the specialized SIC can be omitted unless it has special restrictions. The specialized SIC's own assertions can only refine its parent SIC's assertions, but not overwrite them. For example, suppose that the database designer defines *SIC-1* for *Employee.Salary* on update: "(*Employee.Salary* > *1000*) $\wedge$ (*Employee.Salary* < *1000000*)". The SIC representation for *Manager.Salary* on update is not needed unless there are special restrictions (e.g., "(*Manager.Salary* > *20000*)").

SIC Association A SIC is a set of other SICs if conceptually the enforcement of the **set-SIC** is the same as the enforcement of all of its **member-SICs** and nothing more. The violation action of the *set-SIC* is dummy because if an object violates any *member-SIC*, the violation action of that *member-SIC* will be taken. The certainty factors of these *member-SICs* and *set-SIC* are the same. Thus, we can use a special logical predicate (e.g., *checkmemSIC*, see Section 4.2) to refer to all its *member-SICs* by their names in the *set-SIC* and avoid having to write the same representations of the *member-SICs* explicitly for the object asserted by the *set-SIC*.

The concept of SIC association may be useful for SIC enforcement in a DBMS. For example, all SICs for the same object on the same operation may be grouped as a *set-SIC* or several *set-SICs* according to their certainty factors and/or scheduling requriements. This kind of *set-SIC* is not a new type of SIC. However, in SIC

[1] For example, *checkcomSIC ("Entity*.Attribute*-U-DomNull", EntAttOcc)* could be used to refer to a *component-SIC* called *"Entity*.Attribute*-U-DomNull"* (see Section 4.2) for restricting the attribute *EntAttOcc* not-null.

specifications, conceptually, a SIC on updating the primary key of an entity (or relationship, or relation) can be simulated as two *set-SICs* that refer to SICs for deleting the corresponding old entity (relationship, or relation), and inserting a corresponding new entity (relationship, or relation), respectively.

Advantages of SIC Abstractions The benefits of incorporating SIC abstractions are the following:

- **Facilitating database design:** SIC abstractions permit the database designer to suppress the attention to common SICs and to emphasize those SICs specially pertinent to the object. For example, by SIC aggregation, the database designer need not specially consider domain constraint on insertion of an entity. He (she) can pay attention to others. Similarly, by SIC specialization, while specifying SICs for an entity subtype (relationship, or relation), the database designer need not specify SICs of its super-type that are also relevant to it. The number of explicit SIC representations is thus reduced.

- **Helpful for SIC verification:** It is necessary to verify a set of SICs for nonredundancy and consistency. Redundancy occurs in a set of constraints if some constraints subsume other constraints. Constraints are consistent if there exists a batabase state or a state transition that is allowable with regard to all of the restrictions. SIC specialization is useful for verification since a specialized SIC can only refine its parent SICs, but not overwrite them.

- **Useful for SIC organization and management:** SIC abstractions allow us to organize the SICs in a hierarchy. It also facilitates SIC maintenance and enforcement (e.g., by SIC association).

4 Generic SICs

By applying the above abstraction concepts, we can reduce the number of SICs that must be specified explicitly using the full six components. The concept of generic SICs is introduced to reduce the number of required explicit representations even further. Assume we have the following *generic object types*:

– (1) *Entity** is the generalization (i.e., union) of all entity types that are defined by the database designer. *Entity*.Attribute** is the generalization of all attributes of all entity types that are defined by the database designer.

- (2) *Relationship** is the generalization of all relationship types that are defined by the database designer. *Relationship*.Attribute** is the generalization of all attributes of all relationship types that are defined by the database designer.

These generic object types are conceptual modelling objects. They do not actually exist in a database. That is, neither data definition operations nor data manipulation operations will actually be applied to them. However, we can write some common SIC types (e.g., domain constraints) for them. These SIC representations for generic object types can be called generic SICs (SICs for specific object types can be called *specific SICs* in contrast).

The precondition component of a generic SIC includes logical predicates to indicate clearly the contexts where the SIC type is applicable (e.g., the fact that two entity types are exclusive) and/or identify the related information (e.g., its primary key, etc.) of the object type for which the SIC type applies. These logical predicates contain some uninstantiated variables. For example, *Primary_Key* is a variable in the logical predicate *entity ("Entity*", Primary_Key,...)*. These variables will be instantiated when a specific entity type (e.g., *Employee*) has the constraint information to satisfy the precondition and inherit the SIC. (For example, the above *Primary_Key* can be instantiated to be "EmpId" if the database designer has specified *Employee* as an entity type with the primary key "EmpId", that is, the assertion *entity ("Employee", "EmpId",...)* has been given. Most SIC types (e.g., two entity types are exclusive) only apply to some specific object types. The precondition component of such a generic SIC indicates the conditions for the specific object type where the SIC type is applicable. A few SIC types (e.g., domain constraints) are common to all entity types. In that case, the precondition component of a generic SIC is used to indentify the relevant object in this contex so that its variables can be instantiated with proper values when a specific object type inherits the SIC.

Suppose that we keep the constraint information for specific object types as logical predicates in the database (e.g., in the data dictionary). Also suppose that the DBMS would support SIC inheritance properly. Since all object types defined by the database designer are sub-types of these generic object types, if they satisfy the preconditions of some generic SICs, these SICs would be applied to them by the principle of SIC specialization. Thus, these generic SICs serve as "templates" for common SIC representations and are expected to be inherited by specific object types. We would need <u>only one</u> representation for each SIC type (e.g., two entity types are exclusive or domain constraints) in a database <u>regardless of the number</u> of specific object types to which the SIC type applies.

The idea here is similar to the following simple case. In a database, there are entity types *Employee, Customer, Supplier,* etc. Though the *Person* entity type does not actually in the database, we can write some SICs for Person, which would be inherited by the specific entity types (e.g., *Customer*). The level of our *Entity** type is still higher than *Person.*

The advantage of this approach is to reduce the possibly huge number of explicit representations of SICs so that the conceptual structure and future maintemamce (management) of SICs become easier. In addition, since generic SICs can be pre-defined in an automated database design system, the consultation to elicit SICs would be more efficient.

4.1 Usefulness of the Representation Model

The reasons of why the SIC Representation model is introduced in Section 1.1 are obvious now. It is necessary to identify the object and operation type when applying SIC aggregation or specialization abstractions. The explicit component separation in the SIC Representation model helps identify these components easily. In addition, the representation of generic SICs relies heavily on the precondition component. Most SIC types (e.g., two entity types are exclusive) only apply to some specific object types. The precondition component of such a generic SIC indicates the conditions for the specific object type where the SIC type is applicable. A few SIC types (e.g., domain constraints) are common to all entity types. In that case, the precondition component of a generic SIC is used to identify the relevant object in this context so that its variables can be instantiated with proper values when a specific object type inherits the SIC.

4.2 Representation of Generic SICs

This sub-section introduces the representation of generic SICs for some common SIC types. It is assumed that the database has been designed using E-R model. Therefore, the generic SICs are also represented in terms of E-R schema. The corresponding ones in relational model can be obtained by transformation (references to [50,51]) and are not described here. In all of the following cases, the principle of SIC specialization allows us to omit the explicit SIC representations for specific object types if we have the representations for generic object types.

Domain constraint Representation using SIC Aggregation and Specialization By the principle of SIC aggregation, domain constraints on insertion of an entity/ relationship occurrence can be simulated by applying the domain constraints of all its attributes. There are three separate sub-SICs for restricting not-null, uniqueness, and nonvolatility, respectively, and another sub-SIC dealing with data-type, format, and value. For restricting the insertion of an entity, we need one sub-SIC, which would call the above related sub-SICs (except for the nonvolatility) for asserting attribute domain constraints. In total, the five sub-SICs are sufficient to represent all domain constraints regardless of the number of entity types and their attributes in a database! The number of SICs that must be specified explicity is dramatically reduced through using the SIC abstraction concepts. Similary, five sub-SICs are needed for relationship types and their attributes in a database.

One sub-SIC for restricting not-null is listed here for illustration. Here, the predicate *attribute(Entity/Relationship_Type, Attribute_Name, Domain_Name, Special_Value_Range, Null?, Unique?, Candidate_Key_Attribute?, Changeable?)* is used to specify information about an attribute of an entity or relationship type. Each attribute in the database has such a logical predicate that is stored somewhere, e.g., in the data dictionary. *Domain_Name* is its value domain definition. *Special_Value_Range* is its specific value range information (e.g., *Employee.Salary* has the domain of money with the specific value range between $2,000 and $100,000). The four binary variables, *Null?*, *Unique?*, *Candidate_Key_Attribute?*, *Changeable?* indicate whether the attribute is allowed to be null, unique, a part of a candidate key, and changeable. The *is_not_null (x)* predicate is evaluated to be true if and only if *x* is not "null".

```
Entity*.Attribute*_U_DomNull
     CERTAINTY   certain
     FOR         Entity*.Attribute*
     ON          update
     IF          attribute ("Entity*", "Attribute*", Domain, SpecialVRange,
                    Null?, Unique?, Key?, Changeable?),
                 Null=no
     ASSERT      is_not_null (entity*.Attribute*)
     ELSE        reject
```

This SIC will be inherited by a specific attribute of an entity type, which *Null?* has been specified to be "no". It will be checked on updating the attribute to disallow the new value to be null. We only need one such a generic SIC representation no matter how many attributes in the database have such a restriction.

The representation of generic SIC for domain constraints on insertion of an entity occurrence is not listed here. Rather, its brief description is given as follows. It includes a "set" predicate to retrieve all its attribute names. Then, for each attribute value, three proper sub-SIC names (e.g., *Entity*.Attribute*_U_DomNull* as above) are constructed, and sequentially three *checkcomSIC* (*Component_SIC_Name, Checked_Occurrence*) predicates are used to call proper sub-SICs (one of them is as above) to check whether *Checked_Occurrence* satisfies the restrictions: (1) not-null, (2) uniqueness, and (3) lawful data-type, format, and value. If any of these assertions is not satisfied, the insertion is rejected.

Primary Key Constraint Representation using SIC Association and Specialization
In the traditional E-R model and relational model, there is no internal identifier (*surrogate*) to represent an occurrence of an entity, relationship, or relation. Rather, some attribute or combination of attributes is used as the primary key. Unfortunately, this overloading causes the semantics implied by an update of a primary key to be ambiguous — it may imply a simple update of an attribute or it may imply the deletion of an "old" entity (relationship or tupe) followed by an insertion of a "new" one. Without further information, the SICs related to deletion and insertion must be enforced.

Therefore, a number of SICs must be specified to capture the possible inconsistent states of a database when updating a primary key. The SIC association and specialization concepts will be used ot reduce the number of explicit SICs required. During the database design phase, if a SIC is identified for the insertion (or deletion) of a relationship and there should be a sub-SIC for checking the update of a part of its key, its SIC name is added into an associated *SIC_Name_Set* of the affected key attribute. The *SIC_Name_Sets* of key attributes of a relationship type may be different. However, in the case of a SIC that is relevant to the insertion (or deletion) of an entity type, all affected key attributes of the entity have the same *SIC_Name_Set.*

For example, suppose that we have a SIC such as *"if an employee is assigned to a project, he (she) must participate in an insurance plan"*, and assume the key of the *Employee* and *Project are EmpId* and *ProjId*, respectively. The involved two relationship types are *Assigned_to* connecting *Employee* to *Project*, and *Insure* connecting to *Employee* to *Insurance*. The name of the sub-SIC restricting an insertion of the relationship *Assigned_to* would be inserted into the *SIC_Name_Set* of a special logical predicate (*associated_PKSICs_I*) for the key attribute *Assigned-to.EmpId.* The general format of this special predicate is as follows:

associated_PKSICs_I (Relationship_Type, PKAtt, SIC_Name_Set)
For example, *associated_PKSICs_I ("Assigned_to", "EmpId", SIC_Name_Set)*
Note that since the non-sharing entities are not concerned, the update of another key attribute *Assigned_to.ProjId* of the relationship is not restricted. Similarly, the name of another sub-SIC restricting a deletion of the relationship *Insure*, is inserted into *associated_PKSICs_D* for *Insure.EmpId*. The general format of this special predicate is as follows:
associated_PKSICs_D (Relationship_Type, PKAtt, SIC_Name_Set)
For example, *associated_PKSICs_D ("Insure", "EmpId", SIC_Name_Set)*

Two *set-SICs* are needed for the key attributes of *Relationship**. They are listed here for illustration. By the principle SIC association, the enforcement of such a *set-SICs* is the same as the enforcement of its all *member-SICs*.

Though these two SICs seem to be for any attribute of any relationship, only those having primary key SICs (i.e., having *associated_PKSICs_D* or *associated_PKSICs_I* information) are relevant. In either SIC, the "set" predicate is used to retrieve all member SIC names that belong to *SIC_Name_Set*. Then, the predicate *checkmemSIC (Member_SIC_Name, Checked_Occurrence)* is used to call each member SIC to check whether *Checked_Occurrence* satisfies it. The violation action of the member SIC will be taken if the *Checked_Occurrenece* violates it. The *new/old* function references the new/old value of the referenced object after/before checking the SIC.

```
Relationship*.Attribute*-U-PrimaryKeyDel
    CERTAINTY   certain
    FOR         Relationship*.Attribute*
    ON          update
    IF          associated_PKSICs_D ("Relationship*", "Attribute*", SIC_
                Name_Set)
    ASSERT      set {SIC_Name | belongs-to (SIC_Name, SIC_Name_Set)},
                checkmemSIC (SIC_Name, old (Relationship*))
    ELSE        reject

Relationship*.Attribute*-U-PrimaryKeyIns
    CERTAINTY   certain
    FOR         Relationship*.Attribute*
    ON          update
    IF          associated_PKSICs_I ("Relationship*", "Attribute*",
                SIC_Name_Set)
ASSERT          set {SIC_Name | belongs_to (SIC_Name, SIC_Name_Set)},
                checkmemSIC (SIC_Name, new (Relationship*))
ELSE            reject
```

Similarly, there are two *set-SICs* for key attributes of *Entity**.

Other Common SIC Representation using SIC Specialization A number of other common SIC types can be similarly represented by using generic SIC "templates". Usually, these SICs types can be described in the "closed form" of predicates, that is, without further arbitrary restrictions. If a DBMS finds the related information on a specific entity, relationship or attribute type, e.g., *symmetric* (*Married-to*) indicating its symmetry, the related sub-SICs would be automatically inherited from the generic types.

In [50] generic SICs for twenty-five common SIC types are listed. Generic SICs for two SIC types are described here for illustration. The first is **Absolute Maximum Cardinality Constraint of an Entity Type**, the second is **Incidence Constraint.** It is assumed that DBMS stores the following predicate information for each entity or relationship type.

- The predicate *entity(Entity_Type, Primary_Key, Composite_Key_Set, Absolute_Max_Cardinality)* is used to describe *Entity_Type*. *Primary_Key* specifies its primary key. *Composite_Key_Set* is a set comprised of all composite keys (including primary and non-primary composite keys). *Absolute_Max_Cardinality* specifies the maximum number of occurrences that are allowed in the *Entity_Type.*

- The predicate *relationship_participant(Relationship_Type, Entity_Type, Min_Cardinality, Max_Cardinality)* specifies *Relationshiv_Type's* participant, *Entity_Type*, and the usual relationship cardinalities relative to it. That is, *Min_Cardinality* and *Max_Cardinality* specify the maximum and minimum number of occurrences of the *Relationship_Type* in which each *Entity_Type* in which each Entity-Type occurrence is allowed and is required to participate, respectively. For example, an employee is allowed to participate in 10 projects and is required to participate in at least one.

A subscript notation is used to simulate the "cursor" in Date's UDL [8] to emphasize that a SIC is <u>enforced</u> on an <u>occurrence</u> although it is <u>specified</u> intensionally for an entity/relationship/relation type. Since a SIC is enforced on an occurrence, if there are other occurrences of the same entity, relationship or relation type referred in the precondition or predicate component, different subscripts will be used to distinguish them. The one to be checked is referenced by attaching a default subscript 0, e.g, E_0. Variables with subscripts other than 0 (e.g., E_1) represent any occurence of the same object (e.g., *E*) type. (If there is only one occurrence (i.g., the one to be checked) referred in the precondition and predicate components, the default subscript 0 is omitted.)

In addition, the following manipulation predicates are used in these two examples:

- The predicate *"rship_occ_part (R, Role Type, E)"* is used to evaluate whether an entity occurrence *E* participates in a relationship occurrence *R* with the *Role-Type.*

- The predicate *ent_occ (Entity_Type, E)* is used to evaluate whether an *E* is an occurrence of *Entity_Type,* or used to fetch any occurrence *E* from *Entity_Type* depending on its variable instantiation.

- The predicate *rship_occ (Relationship_Type, R)* is used to evaluate whether *R* is an occurrence of *Relationship_Type,* or used to fetch any occurrence from *Relationship_Type* depending on its variable instantiation.

Absolute Maximum Cardinality Constraint of an Entity Type An absolute maximum cardinality constraint of an entity type restricts the maximum number of occurrences of an entity type that can exist in a database. For example, we might only allow 10000 students or 10 departments in a database. If it is infinite, it is not a restriction. The notation "*" is used to denote either the infinite, or the case that it is not infinite in the mathematical sense, but there is no restriction on the maximum cardinality.

Only one generic SIC representation is needed to for all entity types regardless of the actual number of entity types in the database! The *count* function calculates the number of existing occurrences (including the one to be inserted) of the specified entity type in the database. This number must be less than or equal to the maximal cardinality.

```
Entity*-1-AbsCard
    CERTAINTY   certain
    FOR         Entity*
```

```
ON          insertion
IF          entity ("Entity*", Primary_Key, Composite_Key_Set,
                Abs_Max_Card),
            Abs_Max_Card ≠ "*"
ASSERT      ∃ Entity*_1,
            count (Entity*_1) ≦ Abs_Max_Card
ELSE        reject
```

Incidence Constraint Incidence constraints are very fundamental in the database though only few commercial DBMS enforce them. An incidence constraint requires that the existence of a relationship occurrence always depends on the existence of the participating entity occurrences. That is, a relationship occurrence is allowed to exist only if the participating entity occurrences exist.

Two sub-SICs are needed for this SIC type. The first is to assert that while inserting a relationship occurrence, its participated entity occurrences must exist; otherwise, the insertion is rejected or the system tries to insert some entity occurrences that participate in this relationship. The second is to assert that while deleting an entity occurrence, it must not still participate in any relationship occurrence; otherwise, the deletion is rejected or the system propagates to delete its related relationship occurrence.

Relationship-I-Incidence-(EntType)*

```
CERTAINTY   certain
FOR         Relationship*
ON          insertion
IF          relationship-participant ("Relationship*", EntType,
                Min_Cardinality, Max_Cardinality)
ASSERT      ∃ EntOcc, ent_occ (EntType, EntOcc),
            rship_occ_part (Relationship*, EntType, EntOcc)
ELSE        reject or propagate (insert (EntType, EntOcc))
```

Entity-D-Incidence- (RshipType)*

```
CERTAINTY   certain
FOR         Entity*
ON          deletion
IF          relationship-participant (RshipType, "Entity*",
                Min_Cardinality, Max_Cardinality)
ASSERT      ¬( ∃ RshipOcc, rship_occ (RshipType, RshipOcc),
                  rshipₒcc_part (RshipOcc, "Entity*", Entity*))
        ELSE      reject or propagate (delete (RshipOcc))
```

A DBMS should include an *integrity maintenance subsystem* to enforce SICs. The number of such generic SICs stored in a database would depend on the complexity of the application.

5 Conclusions

This paper claims that database abstractions should consist of data abstractions and SIC abstractions. Data abstractions have been reviewed and clarified. The abstraction concept has been applied to SICs to have SIC abstractions – aggregation, specialization and association. All aspects of data abstractions can not be directly applied. There are new interpretations in terms of the characteristic components of SIC representations. One practical implication of these new SIC abstractions is to facilitate SIC design (specifications), verification, organization and maintenance.

A further research area is to design an automated database design system to assist the database designer to incorporate the necessary SICs and represent them using the above SIC abstractions in a database schema. It would be an expert system with rich heuristics and sophisticated algorithms to elicit SICs and transform them into suitable representations.

References

1. Al-Zobaidie, A., and Grimson, J.B., "Expert Systems and Database Systems: How Can They Serve Each Other? ", *Expert Systems,* Vol. 4, No.1, february 1987, pp. 30-37.
2. Al-Zobaidie, A. and Grimson, J.B., "Use of Metadata to Drive the Interaction Between Database and Expert Systems," *Information and Software Technology,* Vol. 30, No.8, October 1988, pp. 484-496.
3. Beeri, C. and Milo, T., "A Model for Active Object Oriented Database," in *Proceedings of the 17th International Conference of Very Large Data Bases,* Barcelona, September 1991, pp. 337-422.
4. Bic, L. and Gilbert, J., "Learning from AI: New Trends in Database Technology," *Computer,* March 1986, pp. 44-54.
5. Brodie, M.L., "Association: A Database Abstraction for Semantic Modelling," in Chen, P.P. (ed.), *Entity-Relationship Approach to Information Modelling and Analysis (the Second International Conference on E-R Approach),* North-Holland, Amsterdam, 1983, pp. 577-602.

6. Brodie, M.L., "On the Development of Data Models," in Brodie, M.L., Mylopoulus, J., and Schmidt, J.W. (eds.), *On Conceptual Modelling*, Spring-Verlag, Berlin, 1984, pp. 19-47.
7. Charkravarthy, S. and Mishra, D., "Snoop: An Expressive Event Specification Language for Active Databases," *Data & Knowledge Engineering*, Vol.14, 1994, pp. 1-26.
8. Date, C.J., *An Introduction to Database System, Vol. II*, Addison-Westey, Reading, Mass, 1983.
9. dos Santos, C.S., Neuhold, E.J., and Furtado, A.L., "A Data Type Approach to the Entity-Relationship Model," in Chen, P.P. (ed.), *Entity-Relationship Approach to System Analysis and Design*, North-Holland, Amsterdam, 1980, pp. 103-119.
10. Fernandez, E.B., Summers, R.C., and Wood, C., *Database Security and Integrity*, Addison-Wesley, Reading, MA, 1981.
11. Fong, E. and Kimbleton, S.R., "Database Semantic Integrity for a Network Data Manger," in *AFIPS Proceedings of National Computer Conference* California, 1980, pp. 261-268.
12. Frost, R.A., *Database Management Systems*, Granada Publishing Ltd., London, 1984.
13. Furtado, A.L. and Neuhold, E.J., *Formal Techniques for Data Bases Design*, Springer-Verlag, Berlin, 1986.
14. Gallaire, H, Minker, J., and Nicolas, J., "Logic and Databases: A Deductive Approach,", *Computing Surveys*, Vol. 16, No. 2, June 1984, pp. 153-174.
15. Gehani, N. and Jagadish, H.V., "Ode as an Active Database: Constraints and Triggers," in *Proceedings of the 17th International Conference of Very Large Data Bases*, Barcelona, September 1991, pp. 327-336.
16. Goldstein, R.C. and Storey, V.C., "Data Abstraction: The Impact on Database Management," Working Paper, Faculty of Commerce and Business Administration, The University of British Columbia, 1991.
17. Goldstein, R.C., and Storey, V.C., "Materialization," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 6, No. 5, 1994, pp. 835-842.
18. Hull, R. and King, R., "Semantic Database Modelling: Survey, Applications, and Research Issues," *ACM Computing Surveys*, Vol. 19, No. 3, September 1987, pp. 201-260.
19. Jagadish, H.V. and Qian, X., "Integrity Maintenance in an Object-Oriented Database," in *Proceedings of the 18th International Conference of Very Large Data Bases*, Vancouver, British Columbia, Canada, 1992, pp. 469-480.
20. Jarke, M. and Vassiliou, Y., "Coupling Expert Systems with Database Management Systems," in *Artificial Intelligence Applications for Business*, Reitman, W.R. (ed.), Ablex Publishing, Norwood, NJ, 1984, pp. 65-85.
21. Jarke, M. and Vassiliou, Y., "Databases and Expert Systems: Opportunities and Architectures for Integration," in *New Applications of Data Bases*, Gardarin, G. and Gelenbe, E. (eds.), Academic Press, London, 1984, pp. 185-201.
22. Kellogg, C., "From Data Management to Knowledge Management," *Computer*, Vol. 19, No. 1, January 1986, pp. 75-84.
23. Kennedy, A.J. and Yen, D.C., "Enhancing a DBMS Through the Use of an Expert System," *Journal of Information System Management*, Spring 1990, pp.55-60.
24. Lee, K. and Lee, S., "An Object-Oriented Approach to Data/Knowledge Modelling Based on Logic," in *Sixth International Conference on Data Engineering*, California, February 1990, pp. 289-294.
25. Mattos, N.M., "Abstraction Concepts: The Basis for Data and Knowledge Modelling," in

Eighth International Conference on Entity-Relationship Approach, 1988, pp. 331-350.
26. Mees, M. and Put, F. "Extending a Dynamic Modelling Methods using Data Modelling Capabilities: The Case of JSD," in Spaccapietra, S. (ed.), *Entity-Relationship (Fifth International Conference on E-R Approach),* North-Holland, Amsterdam, 1987, pp. 399-418.
27. Missikoff, M. and Wiederhold, G., "Towards a Unified Approach for Expert and Database Systems, in *Expert Database systems (Proc. of the 1st International Workshop),* Kerschberg, L. (ed.), Benjamin/Cummings, Reading, 1986, pp. 383-399.
28. Morgenstern, M., "Active Databases as a Paradigm for Enhanced Computing Environments," in *Proceedings of the 9th International Conference of Very Large Data Bases,* Florence, Italy, 1983, pp. 34-42.
29. Morgenstern, M., Borgida, A., Lassez, C., Maier, D., and Wiederhold, G., "Constraint-Based Systems: Knowledge about Data," in Kerschberg, L. (ed.), *Expert Database Systems (Proc. from the Second International Conference on EDS),* Benjamin/Cummings, Menlo Park, CA, 1989, pp.23-43.
30. Peckham, J. and Maryanski, F., "Semantic Data Models," *ACM Computing Surveys,* Vol. 20, No. 3, September 1988, pp. 153-189.
31. Potter, W.D. and Kerschberg, L., "A Unified Approach to Modelling Knowledge and Data," in Meersman, R.A. and Sernadas, A.C. (eds.), *Data and Knowledge (DS-2),* North-Holland, Amsterdam, 1988, pp. 265-291.
32. Rundensteiner, E.A., "The Role of AI in Databases versus the Role of Database Theory in AI," in *Artificial Intelligence in Databases and Information Systems (DS-3),* Meersman, R.A., Shi, Z., Kung, C-H, (eds.) North-Holland, Amsterdam, 1990, pp. 233-252.
33. Schrefl, M., Tjor, A.M., and Wagner, R.R., "Comparison – Criteria for Smantic Data Models," in *International Conference on Data Engineering,* 1984, pp. 120-124.
34. Shepherd, A. and Kerschberg, L., "Constraint Management in Expert Database Systems," in Kerschberg, L. (ed.), *Expert Database Systems (Proc. form the First International Workshop),* Benjamin/Cummings, Menlo Park, CA, 1986, pp. 309-331.
35. Smith, J.M. and Smith, D.C.P., "Database Abstractions: Aggregation," *Communications of the ACM,* Vol. 20, No. 20, No. 6, June 1977, pp. 405-413.
36. Smith, J.M. and Smith, D.C.P., "Database Abstractions: Aggregation and Generalization," *ACM Transactions on Database Systems,* Vol. 2, No.2, June 1987, pp. 105-133.
37. Stonebraker, M., "The Integration of Rule Systems and Database Systems," *IEEE Transactions on Knowledge and Data Engineering,* Vol. 4, No. 5, October 1992, pp.415-423.
38. Stonebraker, M., Hanson, E.N., and Potamianos, S., "The POSTGRES Rule Manager," *IEEE Transactions on Software Engineering,* Vol. 14, No. 7, 1988, pp. 897-907.
39. Stonebraker, M. and Hearst, M., "Future Trends in Expert Data Base Systems," in *Expert Database Systems (Proc. from 2nd International Conference),* Kerschberg, L. (ed.), Benjamin/Cummings, Reading, 1989, pp. 3-19.
40. Stonebraker, M. and Kemnitz, G., "The POSTGRES Next Generation Database Management System," *Communications of the ACM,* Vol. 34, No. 10, October 1991, pp. 78-92.
41. Storey, V.C., "Understanding Semantic Relationships", *Very Large Data Bases Journal,* Vol. 2, No. 4, 1993, pp.455-488.
42. Storey, V.C., Yang, H.-L., and Goldstein, R.C., "Semantic Integrity Constraints in Knowledge-Based Database Design Systems," *Data & Knowledge Engineering,* Vol. 20, No. 1, June 1996, pp.1-37.
43. Su, S.Y.W. and Raschid, L. "Incorporating Knowledge Rules in a Semantic Data Model:

An Approach to Integrated Knowledge Management," in *the Second Conference on Artificial Intelligence Application,* Miami, Florida, 1985, pp. 250-256.

44. Teorey, T.J., Yang, D., and Fry, J.P., "A Logical Design Methodology for Relational Databases Using the Extended Entity-Relationship Model," *Computing Surveys,* Vol. 18, No. 2, June 1986, pp. 197-222.
45. ter Bekke, J., *Semantic Data Modeling,* Prentice Hall, New York, 1992.
46. Tsichritzis, D.C. and Lochovsky, F.H., *Data Models,* Prentice-Hall, Englewood Cliffs, NJ, 1982.
47. Vassiliou, Y., Clifford, J., and Jarke, M., "How Does an Expert Systems Get Its Data," in *Proc. 9th International Conference on VLDB,* Florence, Oct. 1983, pp. 70-72.
48. Vassiliou, Y., Clifford, J., and Jarke, M., "Database Access Requirements of Knowledge-Based Systems," in *Query Processing in Database Systems,* Kim, W., Reiner, D.S. and Batory, D.S. (eds.), Springer-Verlag, Berlin, 1985, pp.156-170.
49. Widom, J. and Finkelstein, S. J., "Set-Oriented Production Rules in Relational Database Systems", in *Proceedings of ACM SIGMOD International Conference on Management of Data,* Atlantic City, May 1990, pp. 259-270.
50. Yang, H.-L., *Incorporating Semantic Integrity Constraints in a Database Schema,* PH.D. Dissertation, Faculty of Commerce and Business Administration, The University of British Columbia, 1992.
51. Yang, H.-L., "Reformulating Semantic Integrity Constraints Precisely," *Journal of Information Science and Engineering,* Vol. 11, No. 4, December 1995, pp. 513-540.